

Programmierpraxis mit der Java 2 Micro Edition

Im Rahmen des Proseminar „Mobile JAVA (J2ME)“

Sommersemester 2004
Alexander Manz

Inhaltsverzeichnis

1. Einleitung	3
2. Der Aufbau einer Anwendung	4
2.1. Die MIDletSuite	4
2.2. Das MIDlet	4
3. Grafik, Sound und Interaktion	6
3.1. Das Prinzip der Grafikeinbindung	6
3.1.1. Die User Interface Objekte	6
3.1.2. Das Graphics Objekt	7
3.1.3. Das Image Objekt	8
3.2. Audio	9
3.2.1. Audiowiedergabe	9
3.2.2. Ein kleines Beispiel	10
3.3. Interaktion	11
3.3.1. Event - Listener bei UI Komponenten	11
3.3.2. Die Canvas-Klasse zur Interaktion	11
4. Das Beispiel Merlin's Magic Squares	12
4.1. Die Klasse startUp	12
4.2. Die Klasse spielfeld	12
4.2.1. Die Klasse feld	12
4.3. Die Klasse options	12
4.4. Die Klassehelp	12
4.5. Die Ressourcen	13
5. Resumée und Ausblick	14
6. Quellenangaben	14

1. Einleitung

Diese Ausarbeitung beschäftigt sich mit den hauptsächlichsten Prinzipien der Programmierung innerhalb der Java Micro Edition (J2ME). Das Programmieren mit Java ist wohl den meisten bekannt, aber worin liegen nun die Unterschiede zur J2ME, was ist hier besonders zu beachten, wo sind die Grenzen des Möglichen?

Zunächst ist die Micro Edition für Mobile Endgeräte, wie Handys und PDA's ausgelegt. Weil hier die Hardwareressourcen – hauptsächlich bei Handys, PDA's sind hiervon nicht so sehr betroffen - begrenzt sind, muss natürlich auch die Programmiersprache besonders effizient arbeiten.

Die J2ME ist zu diesem Zweck mit den nötigsten und unerlässlichen Befehlen ausgestattet, auf unnötigen Ballast innerhalb der Standardklassen wurde verzichtet. Dies fällt schnell auf, z.B. bei der Suche nach einer Randomfunktion. Im folgenden Beispiel konnte nicht auf die vorgefertigte `Math.random()` Methode zurückgegriffen werden, stattdessen musste eine eigene Methode mit Hilfe des Random-Objektes implementiert werden. Und auch sonst stößt der Programmierer schnell an die Grenzen der J2ME. Doch die Eingeschränktheit bietet durchaus auch ihr Vorteile, so sind die MIDP 1.0 bzw. 2.0 API äußerst übersichtlich und bieten eine gute Möglichkeit sich in die Materie einzuarbeiten.

Im Folgenden wird als Beispiel das Spiel „Merlin's Magic Squares“ ausprogrammiert. Dieses Spiel erschien 1978 auf dem Handheld „Merlin“ von Parker Brothers. Auf diesem Vorläufer des modernen GameBoys konnten neben „Magic Squares“ auch noch 5 weitere einfache Spiele wie „Tic Tac Toe“ oder „Blackjack 13“ gespielt werden.

Das Prinzip von „Merlin's Magic Squares“ ist einfach, auf einer 3x3 Matrix müssen Schalter so angeordnet werden, dass am Ende die acht äußeren auf „An“ (oder *true*) stehen, und der mittlere auf „Aus“ (oder *false*). Dabei beeinflussen sich die Schalter gegenseitig nach einem bestimmten Muster.

Nähere Angaben zum Handheld „Merlin“ und dem mathematischen Prinzip hinter den „Magic Squares“ sind im Internet zu finden:

Handheld: <http://users2.ev1.net/~rik1138/Manuals/ParkerBros-Merlin.pdf>

Math. Prinzip: <http://www.cut-the-knot.org/ctk/Merlin.shtml>

Das Programm: <http://www.uni-ulm.de/1337w4rri0rz/shop/Proseminar.rar>

2. Der Aufbau einer Anwendung

Der erste Schritt beim Programmieren mit der J2ME ist die Auswahl der Programmierumgebung und dem Emulator (im Beispiel das Sun ONE Studio 4 Mobile Edition und das Sun Wireless Toolkit 2.1). Nach dem Start kann dann das Projekt mit dem Anlegen einer MIDletSuite beginnen.

2.1. Die MIDletSuite

Die MIDletSuite beinhaltet am Ende des Programmierprozesses alle wichtigen Klassen inklusive des MIDlets, den Ressourcen wie Bilder, Sounds, etc. und den Informationen über das Programm, gespeichert in einem so genannten Manifest. Alles zusammen wird in einem Jar-File (Java Archive) gespeichert und kann anschließend auf ein mobiles Endgerät übertragen werden.

Im Sun ONE Studio kann eine MIDletSuite einfach über „File“ -> „New“ -> „Templates - MIDP“ -> „MIDletSuite“ erstellt werden. Als nächstes muss ein Name für die MIDletSuite und das Package gewählt. Zu bemerken wäre hier, dass man auch ein existierendes Package auswählen kann, man sollte dann im nächsten Schritt auch das entsprechende MIDlet auswählen. Alternativ kann hier auch ein neues MIDlet erzeugt werden. Beim letzten Schritt wird dann noch der Emulator ausgewählt. Empfehlenswert ist hier, immer mit dem neuesten Wireless Toolkit zu arbeiten.

2.1. Das MIDlet

Ein MIDlet ist das mobile Pendant zum Applet, eine mit der J2ME geschriebene Anwendung die (theoretisch gesehen) auf jedem Endgerät laufen sollte. In der Praxis sind hier aber gewisse Schwierigkeiten anzutreffen, gerade was die Medienunterstützung betrifft. Als größtes Problem ist hier die von Endgerät zu Endgerät unterschiedliche Displayauflösung zu nennen, da sie eine einheitliche Benutzung von Grafiken fast unmöglich macht. Diesem Problem kann zwar in einem geringen Rahmen entgegengewirkt werden, z.B. durch konsequentes Nicht-Benutzen von Grafiken oder dem Bereitstellen von verschiedenen Grafiken für die gängigsten Auflösungen, aber im Großen und Ganzen sind diese Problembehandlungen durch einen hohen Aufwand gekennzeichnet.

Das MIDlet ist das Herzstück einer jeden J2ME-Anwendung, hier liegen die Start-, End- und Pausemethoden und das Display kann direkt angesteuert werden. Ein MIDlet hat drei Zustände: *Active*, *Paused* und *Destroyed*, in denen verschiedene Bedingungen für seine Ressourcen gelten. Der erste Zustand gibt dem MIDlet alle Freiheiten, im Zweiten muss das MIDlet geteilte (shared) Ressourcen freigeben. Dies ist hauptsächlich von Interesse, wenn mehrere MIDlets auf die selben Ressourcen zugreifen können. Im dritten Zustand muss das MIDlet dann alle Ressourcen freigeben und gegebenenfalls Daten noch richtig speichern. Zu jedem dieser Zustände existiert eine Methode, die vom Entwickler überschrieben werden muss. Betrachten wir diese vererbten Methoden:

```
- public void startApp() {}
```

Diese Methode wird aufgerufen, um dem MIDlet mitzuteilen, dass es nun in den Zustand *Active* übergegangen ist. Im Allgemeinen wird hier ein oder mehrere Objekt/e an das Display übergeben. In Nicht-Spiel-Anwendungen kann hier auch, ähnlich wie bei der J2SE/J2EE (Java Standard Edition / Java Enterprise Edition), die gesamte Oberfläche erstellt und an das Display übergeben werden.

- `public void pauseApp() {}`

Diese Methode wird aufgerufen, wenn das MIDlet in den Zustand *Paused* versetzt wird. Hier sollte dann vom Entwickler dafür gesorgt werden, dass er keine wichtigen Informationen verliert, während das MIDlet Pause hat.

- `public void destroyApp(boolean unconditional) {}`

Die letzte der drei Zustandsmethoden wird aufgerufen, wenn das MIDlet beendet werden soll, also in den Zustand *Destroyed* übergeht. Der Entwickler sollte sich an diesem Punkt um die Speicherung von relevanten temporären Daten kümmern, falls er nicht schon zuvor dafür gesorgt hat.

3. Grafik, Sound und Interaktion

3.1. Das Prinzip der Grafikeinbindung

Wie schon von der J2SE/J2EE her bekannt, arbeitet man bei der J2ME mit der Grafikklassse Canvas. Diese ist aber im Gegensatz zur großen Schwester recht umfangreich, da auch Methoden für die Interaktion enthalten sind. Hierzu jedoch später mehr, im Folgenden wird zunächst nur auf die Grafikeinbindung eingegangen.

3.1.1. Die User Interface Objekte

Die J2ME verfügt, wie alle Java Versionen, über vorgefertigte Steuerungselemente, so genannte User Interface Objekte. Einige verfügen über eigene Eingabefunktionen für Text und Zahlen, andere sind nur für die Ausgabe von Informationen zuständig.

Im MIDP 2.0 sind folgende Objekte zur Eingabe fähig:

- `javax.microedition.lcdui.Choice`
Ein Objekt um verschiedene Auswahlmöglichkeiten anzuzeigen. Es ist mit einer Combobox vergleichbar, und kann Strings oder Images enthalten.
- `javax.microedition.lcdui.Command`
Ist in der J2ME das Pendant zum Button der J2SE/J2EE. Er enthält einen String der am Display angezeigt wird und kann verschiedene, bereits vorgefertigte Methoden, wie BACK oder CANCEL, oder selber erstellte Methoden enthalten.
- `javax.microedition.lcdui.DateField`
Dieses Objekt ist für kalendarische Eingaben zuständig. Es kann Eingaben für Termine (im Format TT.MM.JJJJ), Termine und Zeit (im Format TT.MM.JJJJ und HH.MM) oder nur für Zeit (im Format HH.MM) verarbeiten.
- `javax.microedition.lcdui.TextField`
Wie oben schon angedeutet, ist das Textfield für die Eingabe von kurzen Wörtern ausgelegt, da es nur eine Zeile ist. So kann hier z.B. eine Telefonnummer oder ein Name eingegeben werden.
- `javax.microedition.lcdui.TextBox`
Eine Textbox ist zur Eingabe eines längeren Textes dem Textfield zu bevorzugen, da hier mehr als nur eine Zeile zur Verfügung steht. Zusätzlich kann eine Obergrenze für die einzelebaren Zeichen festgelegt werden.

Als reine Ausgabeobjekte sind folgende Komponenten konzipiert:

- `javax.microedition.lcdui.Alert`
Ein Informationsbildschirm, der je nach Belieben modal und/oder temporär begrenzt sein kann. Er kann dazu benutzt werden um dem Anwender Fehler oder Informationen mitzuteilen.
- `javax.microedition.lcdui.Gauge`
Eine Statusanzeige in Balkenform, dem ProgressBar nicht unähnlich. Wird häufig beim Laden von Daten und Herstellen von Verbindungen gezeigt.

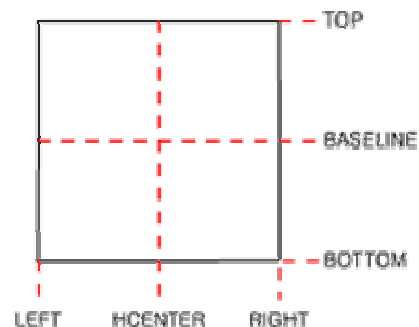
- `javax.microedition.lcdui.Spacer`
Ein nicht zu unterschätzendes Objekt ist der `Spacer`. Er wird benutzt um bei der Darstellung auf dem Bildschirm Zwischenräume zu erzeugen um somit das Design dem Benutzer angenehmer erscheinen zu lassen.
- `javax.microedition.lcdui.Ticker`
Mit dieser Komponente kann der Entwickler einen einfachen Text über den Bildschirm laufen lassen, um z.B. Aktienkurse oder die neuesten Nachrichten bei einer Internetverbindung anzuzeigen.

Im Sun ONE Studio ist zu all den hier genannten Objekten ein sehr gutes Beispiel vorhanden, das sämtliche Komponenten anzeigen kann, so dass man sich ein eigenes Bild davon machen kann. Es ist zu finden im Unterordner `/examples` und hat den Namen `UIDemo.jar`. Wer sich nur das Wireless Toolkit installiert hat, kann das Beispiel unter `/apps/UIDemo` finden.

3.1.2. Das Graphics Objekt

Wichtigstes Hilfsmittel beim Zeichnen mit der `Canvas`-Klasse ist die bekannte `paint(Graphics g)` Methode. Soweit nichts, was der geübte Javaprogrammierer noch nicht kennt. Im Prinzip ist sie wie bei den anderen Versionen von Java aufgebaut, nur sind die Methoden speziell für mobile Endgeräte ausgelegt worden. Die wichtigsten hierbei sind die `draw`- und `fill`-Methoden. Auf einige soll nun genauer eingegangen werden, da sie eher häufig zum Einsatz kommen.

Zuerst aber noch zur Erklärung des Anchor-Systems, das speziell bei der J2ME zum Einsatz kommt. Anchors sind Integerwerte die bestimmen wie ein Objekt an den X-Y-Koordinaten ausgerichtet wird. Zusammengesetzt wird dieser aus je einem der vordefinierten Werte für die horizontale (`LEFT`, `HCENTER`, `RIGHT`) und die vertikale Ausrichtung (`TOP`, `BASELINE`, `BOTTOM`). Beide werden dann durch den bitweisen `or`-Operator kombiniert. Der Zahlenwert Null hat standardmäßig die Bedeutung (`TOP | LEFT`).



Nun zu einer Übersicht der wichtigsten Methoden:

- `void drawImage(Image i, int x, int y, int anchor)`
Wie schon am Name zu erkennen, ist dies die Methode um Bilder auf das Display zu zeichnen. Probleme sollte diese Methode allerdings nicht bereiten, höchstens beim Laden des Images kann es zu Schwierigkeiten kommen, darauf wird im Abschnitt Images aber noch näher eingegangen.
- `void drawRect(int x, int y, int width, int height)`
Die bekannte Methode um einen Rahmen zu ziehen, will man ein gefülltes Viereck zeichnen, sollte man die `fillRect()` Methode nehmen.
- `void drawString(String s, int x, int y, int anchor)`
Diese Methode dient zum Ausgeben eines einfachen Strings auf dem Display. Will man nur ein bestimmter Teil eines Strings ausgegeben, so bietet die J2ME noch die Methode `drawSubstring(String s, int offset, int length, int x, int y, int anchor)`, die den Abschnitt vom Integer `offset` mit der Länge `length` wie gehabt zeichnet.

- `void drawRegion(Image src, int x_src, int y_src, int width, int height, int transform, int x_dest, int y_dest, int anchor)`
 Ab MIDP Version 2.0 bietet die J2ME auch eine Methode um Bilder zu transformieren, wenn auch mit leichten Einschränkungen, so ist es hier nur möglich das Bild in 90°-Schritten zu drehen und/oder zu spiegeln. Zunächst wird mit `Image src` das Bild ausgewählt, mit den Integern `x_src` und `y_src` wird der Ausgangspunkt des zu zeichnenden Ausschnittes gewählt, mit den nächsten beiden, `width` und `height`, wird die Ausschnittsgröße festgelegt. Über den Integer `transform` kann der Programmierer die richtige Transformation definieren. Zur Auswahl stehen die Transformationsinteger aus der `javax.microedition.lcdui.games.Sprite` Klasse:

TRANS_NONE	Keine Transf.	TRANS_MIRROR	Horizontale Spiegelung
TRANS_ROT90	90° Rotation	TRANS_MIRROR_ROT90	Spiegelung dann 90° Rotation
TRANS_ROT180	180° Rotation.	TRANS_MIRROR_ROT180	Spiegelung dann 180° Rotation
TRANS_ROT270	270° Rotation	TRANS_MIRROR_ROT270	Spiegelung dann 270° Rotation

Abschließend werden mit den letzten drei Integeren noch die Ausgabekoordinaten auf dem Display bestimmt.

Eine ausführliche Beschreibung dieser und aller weiteren Methoden findet man in der MIDP-Dokumentation, die man auf der Sun-Homepage herunterladen kann.

3.1.3. Das Image Objekt

Oben wurde bereits das Problem des richtigen Ladens von Image-Objekten angesprochen, dieses kann unter gewissen Umständen zu Verzögerungen bei der Entwicklung führen, da bei den Beispielen, die von Sun mitgeliefert werden, keine genauen Informationen zu finden sind. Deshalb wird das Problem hier eingehend beleuchtet,

Beim Laden aus externen Quellen, wie jpg- oder png-Bildern, kommt es auf den richtigen Pfad an. Arbeitet man mit einer eigenen Programmierumgebung (z.b. Eclipse) und dem Wireless Toolkit, so muss man die Bilder in einem Unterordner des Projektes namens `res` ablegen, dennoch kann man einfach über (`„/Bildname.jpg“`) darauf zugreifen. Benutzt man hingegen das Sun ONE Studio, so muss der Unterordner explizit angegeben werden, da sich Pfadangaben hier relativ zum Arbeitsverzeichnis beziehen.

Bilder werden am über die `create()` Methode des Image Objektes geladen:

```
Image i = Image.create(„/merlin/images/options.png“)
```

Anm.: Dieses Abschnitt beruht auf den Erfahrungen des Autors. Eventuell kann es auf ein fehlerhaftes Plug-In für die Eclipse Entwicklungsumgebung zurückzuführen sein. Auch sei gesagt, dass bei gewissen Versionen im Test beim Laden von Bilder im jpg-Format es zu Problemen kam. Abhilfe schaffte hier die Konvertierung des Bildes ins png-Format.

3.2. Audio

Da heutzutage kaum eine Anwendung, speziell kein Spiel mehr ohne akustische Untermalungen auskommt, bietet auch die J2ME ab MIDP 2.0 die Möglichkeit, Audiodateien wiederzugeben. Dies ist durch das Package `javax.microedition.media.*` gewährleistet. Hier lassen sich diverse Audioformate laden, angefangen bei einfachen Tonsequenzen über die standardisierte Formate Wave Audio Files, AU Audio Files, MP3 Audio Files und MIDI Audio Files, bis hin zu selber erstellten Formaten, die nach MIME Richtlinien definiert sind.

Wave Audio File	<code>audio/x-wav</code>
AU Audio File	<code>audio/basic</code>
MP3 Audio File	<code>audio/mpeg</code>
MIDI Audio File	<code>audio/midi</code>
Ton Sequenz	<code>audio/x-tone-seq</code>

Das System um Audiodateien zu laden besteht aus drei Blöcken. Der erste, überspannende Block stellt der Manager dar, in dem verschiedenen Player geladen werden können. Der zweite Block repräsentiert die Player, die wiederum je eine Audiodatei enthalten. Als letzter Block kommt ein Control-Interface zur Anwendung, das Funktionalitäten zur Lautstärke- und Tonhöhensteuerung bietet.

3.2.1. Audiowiedergabe

Ein Manager kann auf zwei Wegen ein Player-Objekt erzeugen. Der erste Weg ist, dem Player ein Audiofile via Locator-String mitzugeben. Der zweite, etwas kompliziertere Weg ist, zuerst ein InputStream zu erzeugen, dann das Audioformat zu definieren, und abschließend einen Player daraus zu erzeugen.

Der erzeugte Player hat drei einfache Methoden um die Wiedergabe des Audiofiles zu steuern:

1. `javax.microedition.media.Player.start()`
Diese Methode startet das Abspielen des Audiofiles, sollte der Player zuvor mit der Methode `stop()` angehalten worden sein, fährt er an dieser Stelle mit dem Abspielen fort.
2. `javax.microedition.media.Player.stop()`
Pausiert die Wiedergabe des Audiofiles. Hat keine Auswirkung, wenn die Wiedergabe bereits angehalten wurde.
3. `javax.microedition.media.Player.close()`
Schließt den Player und gibt die Ressourcen frei. Auf einen geschlossenen Player kann nicht mehr zugegriffen werden, wenn das Audiofile wieder verwendet werden soll, so ist die `stop()` Methode zu verwenden.

3.2.2. Ein kleines Beispiel

```
try
{
    InputStream is = getClass().getResourceAsStream("start.wav");
    Player p = Manager.createPlayer(is, "audio/x-wav");
    p.start();
}
catch (IOException e)
{
    System.out.println(e.getMessage());
}
catch (MediaException m)
{
    System.out.println(m.getMessage());
}
```

Dieses Codefragment lädt ein Audiofile *start.wav* in einen `InputStream` und erzeugt daraus anschließend einen `Player`, mit der `createPlayer(InputStream i, String type)` Methode. Dabei legt der String *type* fest, dass es sich um ein Wave Audio File handelt. Die `createPlayer()` Methode wirft eine `IOException` und eine `MediaException`, deshalb sollte eine solche Anweisung stets in einem try-catch-Block stehen, damit man etwaige Fehler, meistens fehlerhafte Pfadangaben oder inkompatible Audioformate, schnell erkennen kann.

3.3. Interaktion

Als letzter großer Punkt kommt nun das eigentlich Wichtigste, die Interaktion zwischen Benutzer und Anwendung. Diese kann auf zwei Wegen geschehen, auf der einen Seite über das Event-Listener Konzept, das vor allem bei den UI-Komponenten zum Tragen kommt, auf der anderen über die Grafikkasse Canvas, die über verschiedene Inputmethoden verfügt.

3.3.1. Event – Listener bei UI-Komponenten

Ab MIDP 2.0 verfügen eigentlich nur zwei Elemente der UI-Komponenten über Listener, einmal `javax.microedition.lcdui.Item`, und einmal `javax.microedition.lcdui.Command`. Erstere beinhaltet einen `ItemCommandListener` und einen `ItemStateListener`, Letztere einen einfachen `CommandListener`.

Die beiden `CommandListener` hören auf Events die durch `Commands` ausgelöst werden, der `StateListener` auf Events, die durch einen Zustandsänderung im `Item` ausgelöst werden.

Da das Event-Listener Konzept eigentlich jedem von der J2SE/J2EE her bekannt sein soll, wird in hier nicht näher darauf eingegangen, sondern mehr Augenmerk auf die `Canvas`-Klasse gelegt.

3.3.2. Die Canvas-Klasse zur Interaktion

Wie oben bereits erwähnt, kann die Klasse `Canvas` auch zur Interaktion herangezogen werden, dies ist vor allem bei Spielen und sehr grafiklastigen Anwendungen dem Listener-System vorzuziehen, da die `Commands` im Allgemeinen dem harmonischen Gesamtbild der Anwendung entgegenstehen. `Commands` sind dann vorzuziehen, wenn die Anwendung komplett aus UI-Komponenten besteht, da dann das Design wieder stimmiger wirkt.

Die Klasse `Canvas` kann sämtliche Tasten einer normalen Handytastatur abfangen und darüber hinaus noch Menü- und spezielle Spieltasten simulieren. Das Ganze wird mit Hilfe der Methode `void keyPressed(int keyCode)` realisiert. Hier repräsentiert der Integer `keyCode` eine Taste. Als Erleichterung sind sämtliche Zahlencodes für die Tasten als Konstanten in der Klasse `Canvas` bereits vorhanden, so dass man mit einer einfachen `switch-case` Anweisung eine effektive Tastenabfrage realisieren kann ohne sämtliche Codes als Zahlenwert darzustellen.

Es gibt allerdings zwei verschiedene Darstellungen für die Tasten, einmal als `keyCode` und einmal als `gameAction`. Man kann den Bezeichnungen schon entnehmen, dass `gameAction` wohl speziell für Spiele ausgerichtet ist. Anhand der Tabelle lässt sich das noch verdeutlichen:

keyCode-Wert	Taste	gameAction-Wert	Taste
35	*	1	hoch, 2
42	#	2	links, 4
48	0	5	rechts, 6
49	1	6	unten, 8
50	2	8	fire, 5
51	3	9	game A
52	4	10	game B
53	5	11	game C
54	6	12	game D
55	7		
56	8		
57	9		

Der `keyCode`-Wert ist also eher interessant, wenn man Zahlenwerte und/oder die beiden Zeichen * und # abfragen will, der `gameAction`-Wert ist eher für Richtungsabfragen und Aktionstasten wichtig.

4. Das Beispiel Merlin's Magic Squares

Betrachten wir nun das MIDlet merlin aus dem Beispielprogramm. Die Methoden `pauseApp()` und `destroyApp()` sind beide leer, da es keine relevanten Daten gibt, die gesichert werden müssen, weder beim Eintritt in den Zustand *Paused*, noch beim Zustand *Destroyed*.

Die `ende()` Methode stellt die Funktionalität her, das MIDlet aus dem laufenden Prozess heraus zu beenden. Diese beinhaltet einmal den Aufruf der `destroyApp()` Methode und den Aufruf der `notifyDestroyed()` Methode. Diese Methode benachrichtigt das Managersystem des Endgerätes, dass das MIDlet soeben in den Zustand *Destroyed* übergegangen ist, und somit zu beenden ist.

Die weiteren Klassen des Beispiels sind alle nach demselben Muster aufgebaut, sie sind Erweiterungen der Klasse `Canvas` und sind nach einem bestimmten Muster strukturiert. Zuerst kommen die benötigten Variablen, dann die eigenen Methoden und am Ende die geerbten Methoden und der Konstruktor. Bilder werden bis auf eine Ausnahme immer im Konstruktor geladen, um beim späteren Umschalten zwischen den Klassen eine Verzögerung zu vermeiden. Geerbt wird von der Klasse `Canvas` die `paint()` und die `keyPressed()` Methode.

4.1. Die Klasse `startUp`

Diese Klasse repräsentiert im eigentlichen Sinn den Startbildschirm, sie steuert aber auch die Auswahl der Bildersets für das Spiel selber, sie ist praktisch die Schnittstelle zwischen allen Spielklassen und dem MIDlet. Im Konstruktor finden sich auch Instanzen von allen drei anderen Spielklassen und das MIDlet wird mit übergeben. Dadurch kann direkt auf den Bildschirm gezeichnet werden (über `display`) und die `ende()` Methode kann aus dieser Klasse aus aufgerufen werden.

4.2. Die Klasse `spielfeld`

Diese Klasse (Quellcode auf Seite 16) ist die größte und komplexeste im ganzen Package, da hier die gesamte Spielmethode zu finden ist. Außerdem sind 4 verschiedene Modi zur Tastaturabfrage nötig da verschiedene Zustände erreichbar sind.

Bei den eigenen Methoden ist eine Randomfunktion zu finden, die einen zufälligen Wert zwischen 0 und der übergebenen Zahl zurückliefert. Dies ist nötig, da in der J2ME nicht auf die normale `Math.random()` zurückgegriffen werden kann.

4.2.1. Die Klasse `feld`

Die Klasse `feld` stellt je ein Feld des Spielfeldes dar. Jedes Spielfeld hat einen Zustand `state` der im Konstruktor zunächst auf `false` gestellt wird, später dann direkt, z.B. in der `erstelle()` Methode oder indirekt mittels der eigenen `change()` Methode verändert werden kann.

4.3. Die Klasse `options`

In dieser Klasse kann der Benutzer persönliche Einstellungen treffen, im Moment hauptsächlich sein Bilderset für das Spiel einstellen. Für die kommenden Versionen sind noch weitere Punkte geplant, dazu aber mehr im Ausblick (siehe Punkt 4).

4.4. Die Klasse `help`

In der Klasse `help` (Quellcode auf Seite 27) befindet sich eine kleine Kurzanleitung zum Spiel. Diese besteht aus zwei Bildschirmen, die einfach per Tastendruck weitergeschaltet werden können.

4.5. Die Ressourcen

Im Unterverzeichnis *merlin/images* sind sämtliche Grafiken für das MIDlet zu finden. Abgespeichert wurden die Grafiken im PNG-Format (Portable Network Graphics) nach der Spezifikation 1.0. Teilweise wurden der Alphakanal für Transparenz benutzt, J2ME bietet hierfür auch volle Unterstützung an.

Sämtliche Grafiken sind extra für das Wireless Toolkit 2.1 angefertigt, welches eine Displayauflösung von 180 x 208 Pixel bietet. Bei einer Umsetzung auf ein richtiges Endgerät ist zu beachten, dass die Grafiken angepasst werden.

Die Spielfeldsets bestehen aus zwei Grafiken mit den Namen „feldOn.png“ für ein Feld mit dem Wert *true* und „feldOff.png“ für den Wert *false*. Beide haben ein Format von 32 x 32 Pixel und können Transparenz enthalten. Sie befinden sich in den Unterordnern */merlin/images/set1..8*

Als kleines Gimmick wurde ein Easteregg eingebaut, das ein Wave File abspielt. Dieses befindet sich im Unterordner */merlin/sound*.

5. Resümée und Ausblick

Zusammenfassend ist Programmieren in der J2ME nur eine Gewöhnungssache. Durch den geringen Umfang der API fällt die Einarbeitung recht einfach aus, die Prinzipien der Programmierung bleiben ohnehin die gleichen. Hauptsächlich wurde auf Grafik, Sound und Interaktion mit dem Endgerät Wert gelegt, da in diese vor allem bei Spielen wichtig sind und es in diesem Bereich wohl auch die meisten J2ME Anwendungen gibt. Businessanwendungen werden wohl auch weiterhin eine untergeordnete Rolle spielen. Wer sich dennoch dafür interessiert, dem seien die J2ME-internen Beispiele nahe gelegt. Unter anderem sind hier ein MIDlet für Aktieninteressierte oder ein einfacher Browser aber auch kleinere Spiele zu finden.

Was ist von dem Projekt Merlin noch zu erwarten?

Für eine kommende Version sind einige Erneuerungen geplant, hauptsächlich im Bereich der Einstellungen, so kommt ein Schwierigkeitssystem dazu, das auf den Erkenntnissen der cut-the-knot.org Beschreibung basiert, außerdem sollen eigene Bildersets möglich sein, die ohne große Java- oder Quellcodekenntnisse realisierbar sind. Des Weiteren ist eine bessere Einbindung von Sounds geplant und eine richtige Umsetzung für das Samsung E-700 steht bereits fest.

6. Quellenangaben

Als Quellen kamen nur die API Spezifikationen des MIDP 1.0 und die API Spezifikationen des MIDP 2.0 zur Verwendung

MIDP 1.0 (JSR 37) API Spezifikationen:

<http://jcp.org/aboutJava/communityprocess/final/jsr037/index.html>

MIDP 2.0 (JSR 118) API Spezifikationen:

<http://jcp.org/aboutJava/communityprocess/final/jsr118/index.html>

Beide sind auch über die Sun-Homepage erreichbar

<http://java.sun.com>